Transparent Process Migration: Design Alternatives and the Sprite Implementation

Meenali Rungta
CS 739: Distributed Systems
University of Wisconsin, Madison

27 January 2006
Spring 2006

1 Overview

The paper describes the process migration mechanism used in the Sprite operating system to offload work onto idle machines, and also to evict migrated processes when idle workstations are reclaimed by their owners. The authors have made trade-off between four factors while designing and implementing the mechanism: transparency, residual dependencies, performance and complexity. *The authors claim to have emphasized transparency and performance, but overall the implementation has lead to mix of all above factors being sacrificed for each other at some point, except for transparency. Also, the importance of providing a high degree of transparency at the cost of all other factors and reliability of the system seems less warranted.*

2 Problem Statement and Assumptions

In a network of personal workstations, many machines are typically idle at a given time. These idle hosts represent a substantial pool of processing power that could be harnessed. The authors have implemented the process migration in the Sprite operating system for this purpose.

The assumptions made by the authors about their environment and their workload are listed below:

- Users 'own' their workstations. This means that as soon as the user returns to his workstation, the migrated processes should be automatically evicted. Also, migration itself should not occur automatically to reduce load (as in the processor pool model) but should be user-invoked. Ownership model also implies idle hosts are plentiful, which suggests that sophisticated policies for idle host selection are not required.

- The cluster of machines underlying the system comprises of a few hundered hosts. *However, it is noteworthy that figure 6 in the paper shows that the file-server becomes a bottleneck as 12 hosts performing compilations open files and write back cached object files.*

- The workstations in the Sprite system are diskless, hence the dirty pages of virtual memory and dirty pages in file cache have to be written to the file server at the time of migration.

- The workloads cited in the paper are largely paralled make and simulations. *In particular, network applications and memory-intensive workloads are not included in the performance analysis.*

3 Trade-Offs

3.1 Transparency

Transparency is defined in the paper as:

1. A process's behaviour should not be affected by migration, ie it's execution environment should appear the same and it should produce exactly the same results as it if had not migrated.

2. A process's appearance to the rest of the world should not be affected by migration. In particular, any operation that is possible on an unmigrated process (such as stopping and signalling) should be possible on a migrated process.

Even though tranparent migration gives correct results and makes user's perpective intuitive, it's importance is really workload-dependent.

3.2 Performance

Performance is defined as the efficiency of the act of migration and the efficiency of execution of migrated remote processes. *This dominates the design for virtual memory and open files migration. The VM transfer is done by flushing the dirty pages to the file server and discarding the address space. This makes the act of migration fast. The migration of open files is chosen to be done by transferring the entire state to the remote machine instead of forwarding the kernel calls to it from the home machine. This seeks to avoid the cost of forwarding over the network.*

3.3 Residual Dependencies

A residual dependency is defined as an ongoing need for a host to maintain data structures or provide functionality for the process even after the process migrates away from the host. *The authors leave several residual dependencies for achieving transparency, like in case of 'fork' and 'exit'. The resulting distributed process state degrades the performace and reliability of the system, and also makes the system more complex.*

3.4 Complexity

The complexity of the system can be best measured by it's maintainability. The authors mention that their migration code broken often and easily, due to other changes in the Sprite kernel. Clearly, the implementation turned out to be pretty complex.

4 Techniques

The main techniques employed to achieve migration were:

- Maintaining global state - For example, the authors changed the file system so that every machine in the network sees the same namespace.

- Transferring state from source machine to target - This was used to migrate virtual memory, open files, process identifiers etc.

- Forwarding kernel calls home - The technique had to employed in cases like gettimeofday() and getpgrp(). *It was noted that by using a slight optimization, the authors could have saved themselves the penalty of forwarding gettimeofday(). This would involve forwarding it just once and storing the skew in time on the remote machine. Thereafter the call could be served locally.*

5 Case Studies

5.1 Migrating Virtual Memory

There are three approaches to transfer VM: freezing the process and doing a monolithic transfer, pre-copying the pages and lazy-copying the pages. The authors have chosen a variant of lazy copying - the process freezes on the source machine, its dirty pages are flushed, and the process starts on the target with no resident pages. Paging on-demand is used to build up the address space from scratch on the target machine.

5.2 Migrating Open Files

There are three main components of the state associated with an open file. These are transferred as follows:

- file reference - The open file is opened on the target machine before closing it on the source, to get around the delete-of-an-opened-file semantics of UNIX.

- caching information - Since the file was open on source and target machines simultaneously, as noted

above, it appear to be write-shared and caused the server to disable caching for the file unnecessarily. This was due to the pre-existing consistency semantics of the Sprite file system.

- access position - This becomes shared among machines if one of the processes sharing the access position is migrated. In this case, the authors chose to disable storing of access position on either machine, and forwarding all operations on the file to the server which maintains the position. *In our opinion, this makes all subsequent file operations tediously slow, and seems like an undesirable design choice.*

6 Conclusions

The paper is a step forward from the distributed systems of 1985, which had no idea of targeted workloads and applications, and could never take-off. The authors were, understandably, concerned mainly about designing a system that satisfied their users' needs - compiling sprite kernels and performing simulations. However, these two goals could have been, probably, more easily achieved by a simpler system targeted towards making these two specific applications fast, instead of a full-fledged process migration system.